Team 24

Detailed Design Document

1 Virtual AP Creation Script (create_wlan0_ap.sh)

1.1 Detecting Wireless Hardware and AP Support

The script first identifies the underlying physical wireless interface and verifies that it supports Access Point (AP) mode. The real device name is obtained using `iw`:

Listing 1: Detecting the real wireless interface

```
REAL_DEV=$(iw dev | awk '$1=="Interface"{print $2}' | head -n1)
```

Support for AP mode is checked using:

Listing 2: Checking AP mode support

```
iw list | grep -q "* AP"
```

If AP mode is not supported, the script exits. This prevents misconfiguration on incompat ible hardware.

1.2 Installing Required Services

To provide AP and DHCP functionality, the script installs the following packages:

- `hostapd` for managing the wireless AP, authentication, and beaconing.
- `dnsmasq` for lightweight DHCP and DNS services.

An example installation command is:

Listing 3: Installing hostapd and dnsmasq

```
apt install -y hostapd dnsmasq
```

1.3 Creating the Virtual Interface

The script creates a virtual AP interface named `wlan0` on top of the real Wi-Fi device:

Listing 4: Creating the virtual AP interface

```
iw dev $REAL_DEV interface add wlan0 type __ap
ip addr add 192.168.4.1/24 dev wlan0
```

Here, the ___ap type forces the interface into AP mode. The IP address 192.168.4.1/24 serves as the default gateway for connected clients.

1.4 Hostapd Configuration

The `create_wlan0_ap.sh` script dynamically generates the file `/etc/hostapd/hostapd.conf`. A typical configuration for a 2.4 GHz WPA2-protected AP is:

Listing 5: Example hostapd configuration

```
interface=wlan0
ssid=MyLinuxAP
hw_mode=g
channel=6
wpa=2
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
```

Key parameters:

- `ssid`: network name broadcast by the AP.
- `hw_mode=g`: 2.4 GHz band using 802.11g.
- `channel=6`: operating channel for the AP.
- `wpa_passphrase`: WPA2 pre-shared key for security.

1.5 DHCP and NAT Configuration

To allocate IP addresses, `dnsmasq` is configured with a DHCP range inside the AP subnet, for example:

Listing 6: dnsmasq DHCP range configuration

```
dhcp-range=192.168.4.10,192.168.4.100,12h
```

NAT rules are set up via `iptables` so that clients connected to `wlan0` can access the internet through a wired interface such as `eth0`:

Listing 7: Enabling NAT for internet sharing

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```

This combination of DHCP and NAT transforms the Linux host into a fully functional wireless router for the `wlan0` AP.

2 Hotspot Activation Script (`start_ap.sh`)

The `start_ap.sh` script acts as a reusable entry point to bring up the AP, restart services, and apply runtime parameters such as transmit power and additional hostapd settings.

2.1 Service Management

To avoid conflicts, the script typically stops any existing instances of NetworkManager, dnsmasq, and hostapd that may interfere with manual configuration. It then restarts the required services with the new configuration files.

2.2 Transmit Power and RF Parameters

The script may also adjust the transmit power of the AP interface:

Listing 8: Setting transmit power on the AP interface

```
iw dev $AP_IF set txpower fixed 1000
```

Here, `$AP_IF` is the AP interface (for example, wlan0), and the value is specified in units of 0.01dBm.

2.3 Example Hostapd Parameters

For a specific SSID and channel, the script can generate or modify a hostapd configuration similar to:

Listing 9: Example hostapd configuration in start$_a p.sh$

```
interface=wlan0
ssid=V_Happy_AP
hw_mode=g
channel=1
ieee80211n=1
ht_capab=[HT20]
country_code=US
wpa=2
wpa_passphrase=12345678
```

These parameters define a 2.4GHz AP with 20MHz channel width, HT capabilities, and region-specific constraints.

2.4 Forwarding and Routing Rules

To ensure clients can reach the external network, `start_ap.sh` configures forwarding and firewall rules such as:

Listing 10: Forwarding and firewall configuration

```
iptables -A FORWARD -i $AP_IF -o $NET_IF -j ACCEPT
iptables -A FORWARD -i $NET_IF -o $AP_IF -m state \
    --state RELATED,ESTABLISHED -j ACCEPT
```

Here, `$NET_IF` is the upstream interface providing internet connectivity (e.g., `eth0`).

3 Band Steering Logic (`wifi_band.sh`)

Band steering improves performance by allowing the operator to select between:

- 2.4 GHz band: longer range and better penetration, but typically lower throughput and more interference.
- 5 GHz band: higher throughput and lower interference, but shorter range.

The `wifi_band.sh` script automates modification of band-related parameters in the AP configuration script (`start_ap.sh`) or its generated `hostapd.conf`.

3.1 Parameter Switching

Depending on the command-line argument (for example, 2.4 or 5), the script uses sed to switch hardware mode and channel:

Listing 11: Switching to 2.4 GHz band

```
sed -i 's/hw_mode=.*/hw_mode=g/' $AP_SCRIPT
sed -i 's/channel=.*/channel=3/' $AP_SCRIPT
```

Listing 12: Switching to 5 GHz band

```
sed -i 's/hw_mode=.*/hw_mode=a/' $AP_SCRIPT
sed -i 's/channel=.*/channel=36/' $AP_SCRIPT
```

For 5GHz operation, additional 802.11ac (VHT) parameters may be appended if they are not already present:

Listing 13: Enabling 802.11ac / VHT options

```
echo 'ieee80211ac=1'       >> $AP_SCRIPT
echo 'vht_oper_chwidth=1'  >> $AP_SCRIPT
```

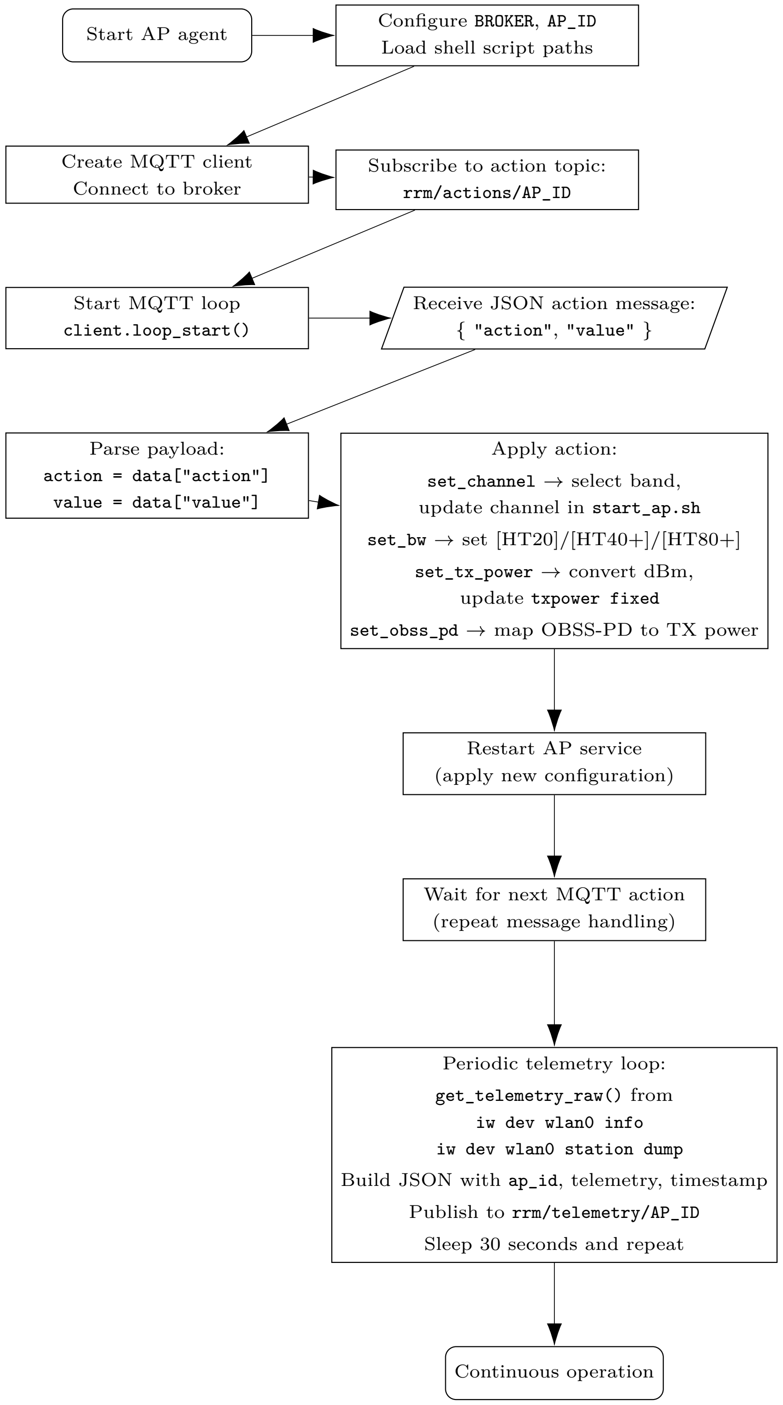
This enables wider channels and high-throughput features on capable hardware.

3.2 Operational Comparison

Table 1 summarises the effective behaviour of selecting each band through `wifi_band.sh`.

Band	Pros	Script Effect
2.4 GHz	Long range, better coverage	`hw_mode=g`, low channel, HT mode
5 GHz	Higher speed, less congestion	`hw_mode=a`, channel 36, VHT/AC enabled

Table 1: Qualitative comparison of 2.4 GHz vs 5 GHz band selection.



Start AP agent
Configure BROKER, AP_ID
Load shell script paths
Create MQTT client
Connect to broker
Subscribe to action topic:
rrm/actions/AP_ID
Start MQTT loop
client.loop_start()
Receive JSON action message:
{ "action", "value" }
Parse payload:
action = data["action"]
value = data["value"]
Apply action:
set_channel → select band,
update channel in start_ap.sh
set_bw → set [HT20]/[HT40+]/[HT80+]
set_tx_power → convert dBm,
update txpower fixed
set_obss_pd → map OBSS-PD to TX power
Restart AP service
(apply new configuration)
Wait for next MQTT action
(repeat message handling)
Periodic telemetry loop:
get_telemetry_raw() from
iw dev wlan0 info
iw dev wlan0 station dump
Build JSON with ap_id, telemetry, timestamp
Publish to rrm/telemetry/AP_ID
Sleep 30 seconds and repeat
Continuous operation

4 Architechtureeal Design of Sensing Orchestrator using GNU radio and SDR.

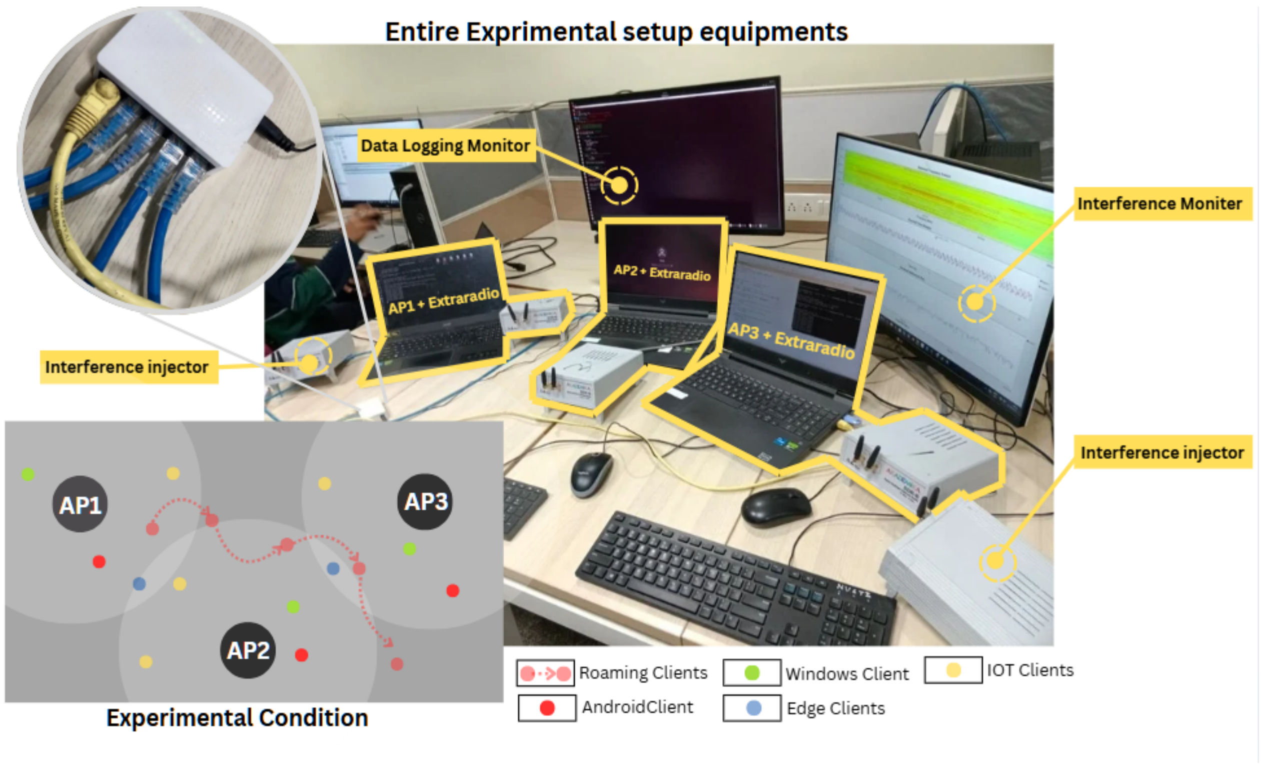


Figure 1:

The complete experimental setup is presented in the figure. Below it, an experimental testbed was implemented to verify our reinforcement learning (RL) approach. The Internet of Things (IoT) clients were constructed using multiple ESP32 microcontrollers, while all other clients comprised mobile phones and laptops.

The Access Point (AP) with an extra radio was created utilizing a Laptop (model: *[Insert Model]*) operating in hostapd mode under Linux, alongside one Software-Defined Radio (SDR). The entire sensing orchestrator was implemented on the SDR Akademika B (Device: *AdamPluto*) using GNU Radio.

The following challenges were faced during the implementation:

Device Sampling Frequency Limitation:

- **Challenge:** The device's stable maximum sampling frequency is approximately 10 MHz. This presented a discrepancy when trying to sense 20 MHz channels.
- **Mitigation:** Since we are not demodulating the signal but rather estimating noise using statistical methods (based on Power Spectral Density and total signal power) instead of equalization, this limitation did not significantly impact the overall outcome.

Intermittent Sensing Operation:

- **Challenge:** The sensing process would halt after about 40 timesteps. In GNU Radio, dynamic sensing requires updating the global variable for the central frequency. Providing "Parent access" to the MAB Scheduling block for this update inadvertently clashed with the return value of the iio libraries.
- **Mitigation:** A custom auto-error mitigation mechanism (`Watchdog.sh`) was implemented. This script automatically restarts the MAB, loads its previous state upon any error or crash, ensuring continuous operation.

Data Bandwidth Bottleneck:

- **Challenge:** The high sampling rate of 10 kHz resulted in a data bandwidth that exceeded the capacity of the USB connection, causing a system bottleneck.

Buffer Overflow and System Stall:

- **Challenge:** As a consequence of the high sampling rate issue, running the system for extended periods (days) caused the buffer to fill up, leading to a complete halt of the SDR.
- **Mitigation:** The `watchdog.sh` script was enhanced to include a scheduler. Every 30 minutes, it reboots the system, flushes the SDR, waits for a successful ping (approximately 15 seconds), and then resumes the sensing operation.

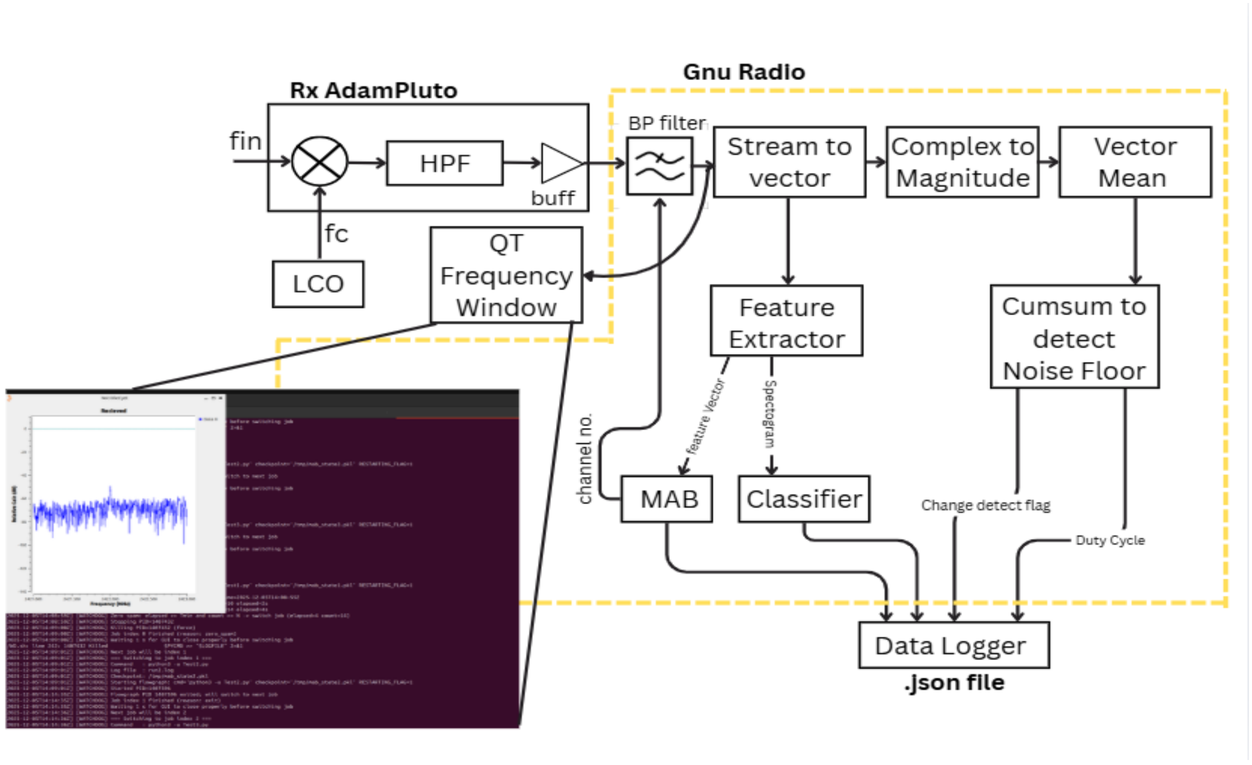
Innovation and Strategy Summary:

Trainable Radial Basis Network Reward Function:
The reward function is based on a trainable radial basis network (RBN) that dynamically adapts to changing network conditions. The network is designed to learn and optimize the reward function based on network parameters such as noise levels. In this setup, higher noise levels lead to a higher reward, allowing the system to prioritize environments where noise is higher, thus improving the robustness and adaptability of the network in fluctuating conditions.

KalmanUCB Bandit:
The KalmanUCB (Upper Confidence Bound) bandit algorithm is used to optimize decision-making processes under uncertainty. It combines Kalman filtering for noise estimation with the Upper Confidence Bound approach for exploring and exploiting actions. This strategy balances exploration and exploitation by adjusting the confidence in estimates and making better-informed decisions over time. It is especially useful in environments with high variability, enabling effective decision-making even in dynamic and noisy conditions.

Kalman Layer for Noise-Based Smoothing:
To handle noise in the environment and smooth out measurements, a Kalman filter layer is integrated into the system. This layer operates as a noise-smoothing mechanism by estimating the true state of the system, filtering out random fluctuations and errors in the measurements. It helps in producing more stable and accurate outputs, improving overall system performance and decision-making processes, especially in scenarios with significant noise or interference.

Final Simulations and Results, and Comparison with Simulation Data:



```
[illegible]
```

Figure 2: SDR Based Hardware Implementation of Dynamic channel sensing and interference classifier

4.1 Total DFS Aware Algorithm for Dynamic Sensing of Channel

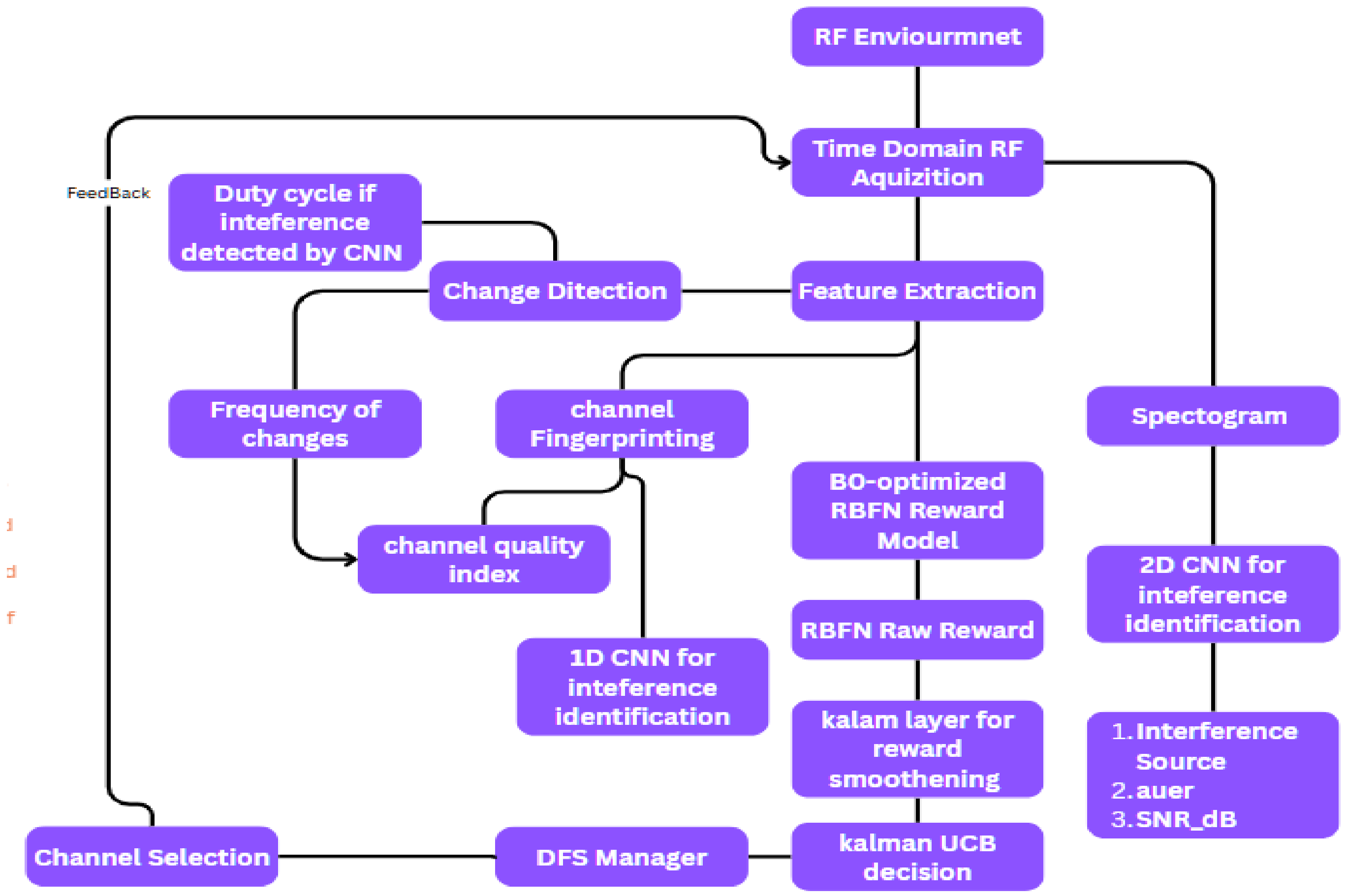


Figure 3: Dynamic Channel scheduling flowgraph with DFS state aware Logic layer

Our sensing orchestrator integrates feature extraction, channel fingerprinting, spectrogram imaging, machine-learned reward prediction, Kalman filtering, and multi-armed bandits into a unified architecture. Scalar OFDM features and fingerprints feed an RBFN reward model and optional 1D-CNNs, while spectrogram snapshots enable 2D-CNN inference. A Kalman filter smooths and predicts rewards under DFS constraints, and Kalman-UCB uses this uncertainty-aware reward to make optimal band decisions. The combination of RBFN prediction, Kalman smoothing, and CNN-based multi-modal learning creates a robust, adaptive, and intelligent spectrum sensing framework capable of operating in highly dynamic OFDM and DFS environments.

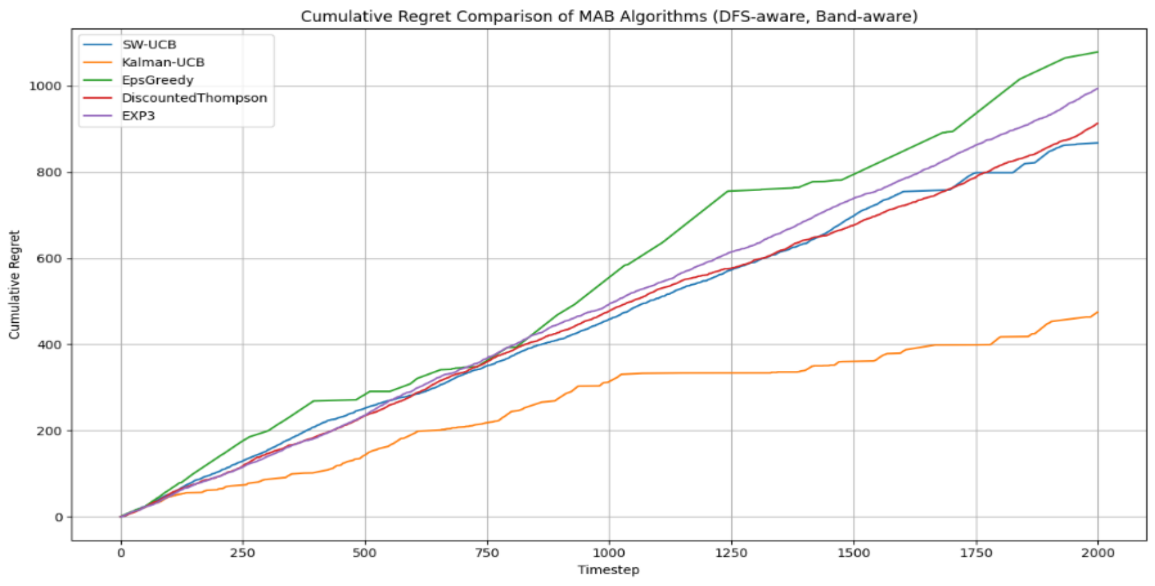


Figure 4: Cumulative Regret Curves for all Bandit algorithms for performance analysis

5 Modifying Transmit Power, Channel, and Bandwidth Parameters in `start_ap.sh`

This report explains the process and purpose of updating Access Point (AP) configuration parameters through automated script editing and restarting the hotspot service.

The following `sed` commands were used to dynamically modify the configuration inside `start_ap.sh` to change transmit power, Wi-Fi channel, and channel bandwidth (HT capability):

```
sudo sed -i ’s/txpower fixed [0-9]\+/txpower fixed 1500/’ start_ap.sh
sudo sed -i ’s/^channel=[0-9]\+/channel=3/’ start_ap.sh
sudo sed -i ’s/^ht_capab=.*/ht_capab=\[HT20\]/’ start_ap.sh
```

These commands update the script at the source level, ensuring that the corresponding parameters in the automatically generated hostapd.conf and AP configuration are applied when the AP restarts.

5.1 Description of Each Modification

Command: `txpower fixed 1500`
Purpose: Update transmit power level
Effect: Attempts to increase signal output strength to improve coverage

Command: `channel=3`
Purpose: Sets AP broadcast channel

Effect: Moves AP to a less congested channel to improve stability

Command: `ht_capab=[HT20]`
Purpose: Sets channel width to 20 MHz
Effect: Improves compatibility and reduces interference

5.2 Need for AP Restart

After each change, the AP is restarted using:

```
sudo ./start_ap.sh
```

Restarting is necessary because:

- The hardware in use did not support live reconfiguration using real-time interface tools such as `iw dev <iface> set channel` or `iw phy set txpower`.
- `hostapd` requires a full restart to reload modified configuration parameters.
- Modifying values directly within `start_ap.sh` ensures that configuration persists.

Thus, the scripts automate changes that would otherwise need manual modifications in configuration files, while the restart applies them immediately.

The use of script-based configuration edits combined with AP restarts provides a practical workaround for hardware that does not support real-time wireless parameter changes. By updating start_ap.sh via sed, the AP can be reconfigured smoothly without manually editing configuration files, ensuring faster testing and deployment workflow.

6 MTT

6.1 Fast-Loop Controller

The fast loop is a short-timescale RRM controller that reacts quickly to interference and load by proposing channel and bandwidth changes per AP.

Input:

- AP telemetry
- Per-channel summaries

Output:

- Action proposals such as:

$$\{ap\_id : "ap3", type : "channel", value : 44, reason : "lowerinterference\&load"\}$$

- Another example:

$$\{ap\_id : "ap1", type : "width", value : 20, reason : "secondary40MHzhalfisbad"\}$$

These proposals are later checked by the event loop and applied to the APs.

6.1.1 Inputs and Warm-Up

On each tick, the fast loop collects per-AP telemetry (current channel, band (2.4/5 GHz), width (20/40 MHz), Tx power, interference estimate, and client list (MAC, RSSI, throughput if available)) and per-channel summaries (interference power in dBm, total client count, and optionally channel utilization). APs/channels with fewer than `FASTLOOP_WARMUP_SUMMARY_COUNT` samples are skipped until enough history is available to avoid unstable decisions.

6.1.2 Channel Health Ranking

For each band (2.4, 5 GHz), the controller computes a health score per channel:

$$\text{score}(ch) = w_{\text{intf}} \cdot \text{interference\_dBm} + w_{\text{load}} \cdot \text{client\_count}$$

Lower score means a better channel (less interference, fewer clients). Channels are sorted best to worst. A set `DFS_CHANNELS` marks DFS-restricted 5 GHz channels to handle radar constraints.

6.1.3 Per-AP Decision Logic

For each eligible AP (not frozen or under maintenance), the fast loop reads its current channel, band, width, interference, and DFS status and applies the following logic:

- **DFS / radar handling:** If the AP is on a DFS channel and `curr_interference_dBm` > `RADAR_THRESHOLD_DBM`, this is treated as radar. Candidates are restricted to non-DFS channels. If the best non-DFS channel improves the score by at least `MARGIN_DB`, the controller proposes a channel change with reason "DFS radar / high interference".
- **40 MHz logic (5 GHz):** If on 40 MHz, the controller compares the primary and secondary 20 MHz halves. If the secondary half is much worse, it proposes narrowing to 20 MHz on the primary. If both halves are poor relative to other channels, it may propose moving the AP to a better 20 MHz channel in the same band.
- **20 MHz logic (2.4 & 5 GHz):** The controller looks at top candidate channels in the same band, only considering channels where interference is at least `MARGIN_DB` lower and `client_count` is not higher than the current channel. If a clearly better channel exists, it proposes a channel change; otherwise it does nothing to avoid flapping.

6.1.4 Guardrails and Safety Filters

Before finalizing actions, the fast loop applies several guardrails:

- **Per-AP cool-off:** If `now - last_change_time[ap_id]` < `COOL_OFF_PERIOD`, new proposals for that AP are dropped.

- **Daily site budget:** A `SITE_CHANGE_LOG` enforces `SITE_CHANGE_BUDGET_PER_DAY` so the network is not over-tuned.
- **Rollout / A/B limits:** Only APs in the allowed experiment set (e.g., canaries) keep their proposals.
- **Sanity checks:** The controller rejects actions that move to disabled channels, violate regulatory rules, or conflict with higher-priority freezes from the slow loop.

6.1.5 Handover to Event Loop

The remaining actions form the fast-loop proposal list. The main event loop merges them with quiet-hours/exam profiles, operator overrides, and KPI guardrails, applies the final actions through the AP driver/API (channel/width changes), and logs each change with timestamp and reason. This design allows the fast loop to be aggressive and interference-aware, while the event loop enforces global safety and policy.

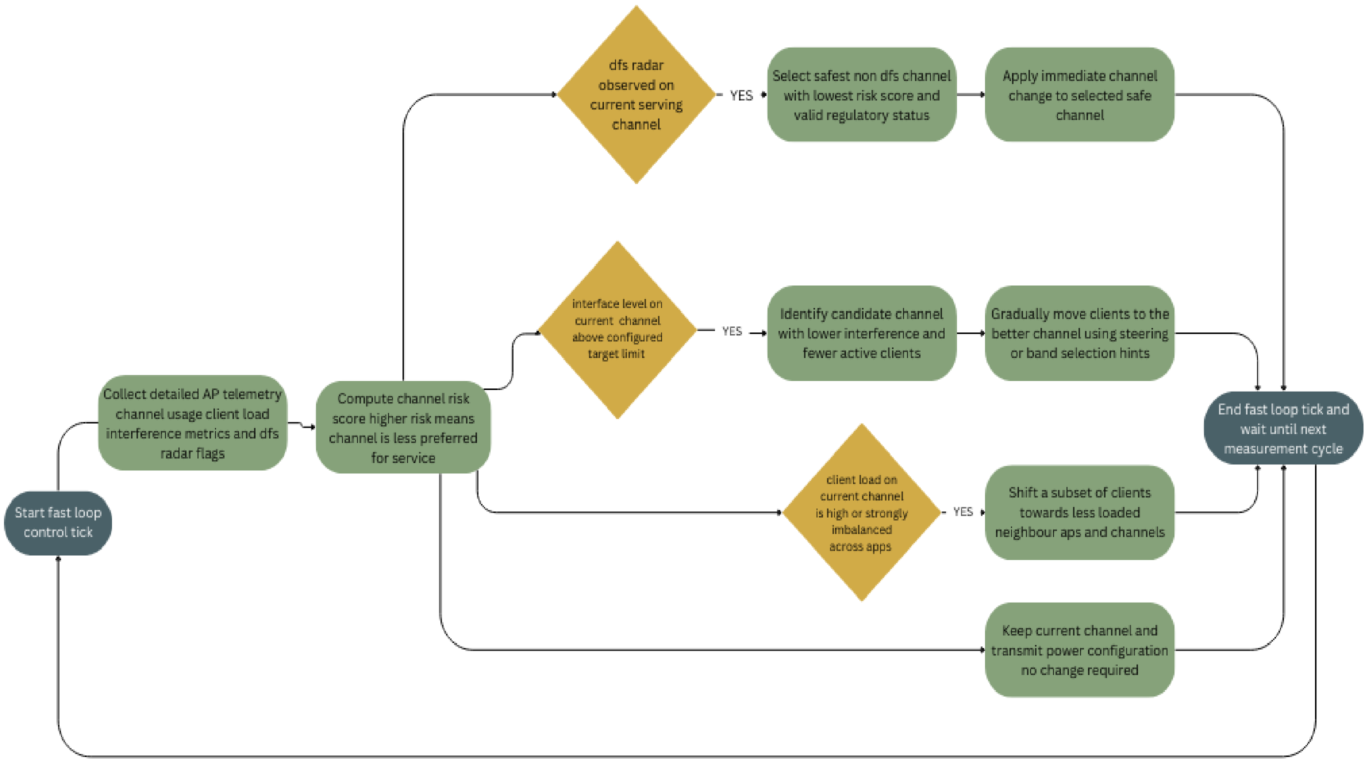

Start fast loop control tick
Collect detailed AP telemetry channel usage client load interference metrics and dfs radar flags
Compute channel risk score higher risk means channel is less preferred for service
dfs radar observed on current serving channel
YES
Select safest non dfs channel with lowest risk score and valid regulatory status
Apply immediate channel change to selected safe channel
interface level on current channel above configured target limit
YES
Identify candidate channel with lower interference and fewer active clients
Gradually move clients to the better channel using steering or band selection hints
client load on current channel is high or strongly imbalanced across apps
YES
Shift a subset of clients towards less loaded neighbour aps and channels
Keep current channel and transmit power configuration no change required
End fast loop tick and wait until next measurement cycle

Figure 5: Fast loop

6.2 Event Loop Design Report

The event loop is a slow-timescale controller that, every `FAST_LOOP_PERIOD`, reads telemetry and schedules and decides per-AP channel, bandwidth, and Tx power changes under strict safety and regulatory guardrails.

Design goals (high level):

- **Stable QoE:** Keep RTT, loss, and coverage within targets.

- **Exam protection:** Prioritize exam/meeting APs during critical slots.
- **Safe control:** Avoid flaps via cool-off, budgets, and KPI checks.

6.2.1 One-Time Initialization

- Load AP inventory + roles (exam_hall, near_exam, canteen, normal).
- Load RF constraints: allowed bands/channels, widths, Tx power ranges, DFS rules.
- Load schedules: exam/meeting calendar, quiet-hour profiles, maintenance/isolation flags.
- Init guardrail state: `SITE_CHANGE_LOG`, `last_change_time[ap_id]`, KPI thresholds, RL/baseline handles.

6.2.2 Per-Tick Flow

Tick Start

Sleep for `FAST_LOOP_PERIOD`, then start a new tick, record time `t` and open a simple log entry {tick_id, sim_time, wall_time}.

Telemetry Snapshot

- Read per-AP telemetry: channel, width, Tx power, client RSSI/RTT bins, interference, neighbours.
- Build per-channel summary: interference dBm, client count, utilization (if available).
- If data is stale beyond a threshold, log "telemetry stale, skipping tick" and exit this tick with no changes.

Schedule Evaluation

- Determine active exam / meeting / quiet windows at time `t`.
- For each AP, derive role (exam_hall, near_exam, etc.) and profile (exam / quiet / default).
- Translate to intents: boost + stabilize exam APs, de-noise neighbours. (Only intents, not actions yet.)

6.2.3 Action Proposals

Fast Loop

Fast loop uses snapshot + channel summary to propose low-latency RRM actions (channel/width changes with simple reasons like "high interference", "load balance").

RL / Baseline

RL/baseline builds features from telemetry and returns per-AP actions; anything violating basic safety is dropped or replaced by baseline.

Schedule-Based

Schedule logic converts intents into concrete RF tweaks: exam APs get safer/higher Tx (within limits) and stable channels; neighbour APs back off to protect exam QoE.

6.2.4 Merge, Guardrails and Commit

Per-AP Proposal Merging
Per AP, proposals are merged with fixed priority:

- 1) Hard safety / manual overrides (maintenance, isolation, frozen).
- 2) Schedule / exam-profile actions.
- 3) RL proposals (post-safety).
- 4) Fast-loop RRM proposals.
- 5) No-change default.

Channel-freeze from schedule cancels all channel moves; illegal RL band/power is clamped or dropped. The result is one candidate config (channel, width, Tx power, reason) per AP.

Per-AP Cool-Off
Block channel/width changes if `now - last_change_time[ap_id]` < `COOL_OFF_PERIOD` to avoid flapping; optionally allow small Tx nudges.

Site Change Budget
Count actions in `SITE_CHANGE_LOG` over the last 24h and enforce `SITE_CHANGE_BUDGET_PER_DAY`; drop non-essential or disruptive changes once the budget is hit.

KPI + Regulatory Checks
Reject proposals predicted to break RTT/loss/coverage limits and ensure all actions respect max Tx power, allowed channels/widths and DFS rules.

Commit and Loop Back
Apply approved changes to APs, update `last_change_time[ap_id]`, append entries to `SITE_CHANGE_LOG` and write a short human-readable summary of active schedules, actions taken and actions blocked by guardrails. Then return to the next tick.

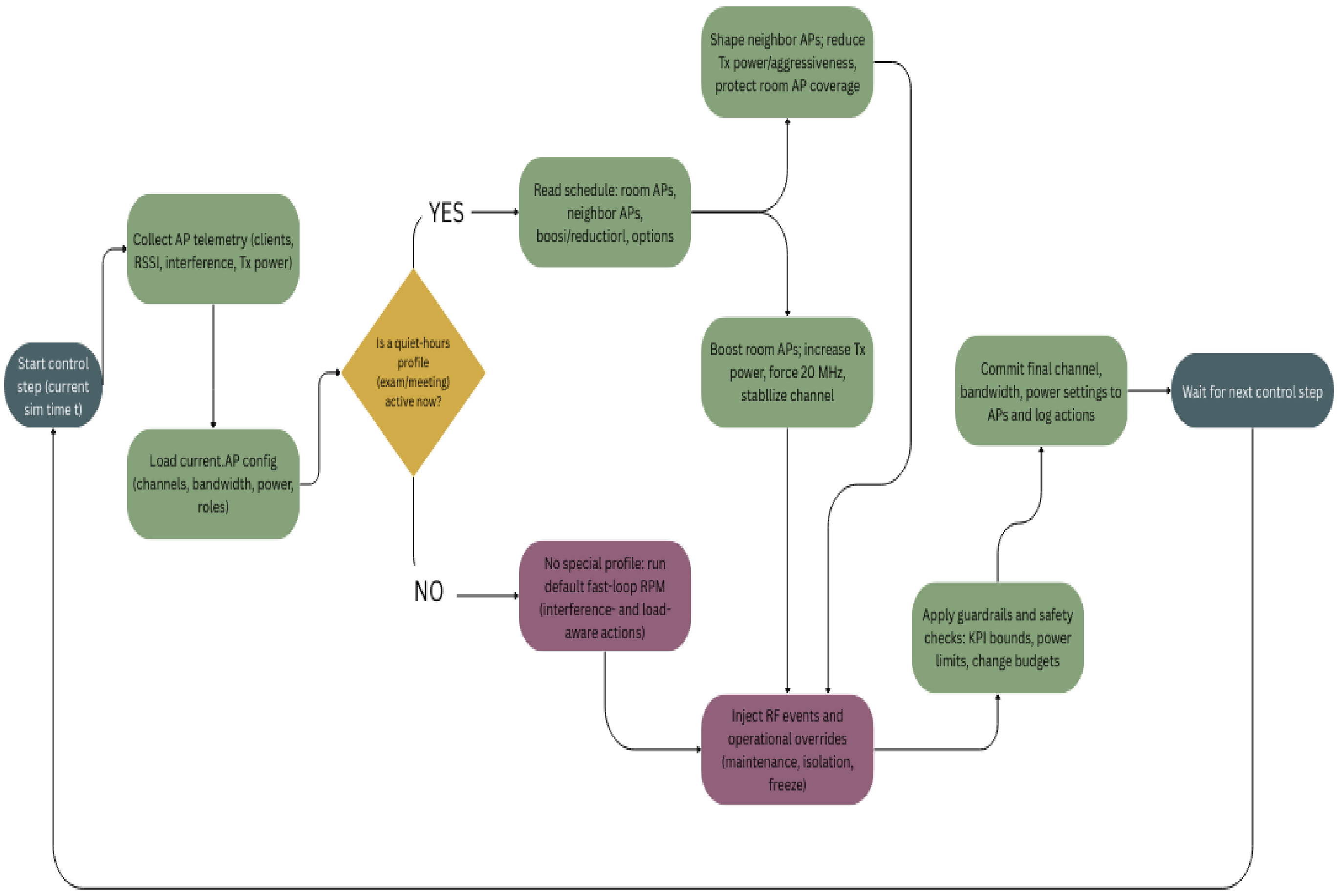

Start control step (current sim time t)
Collect AP telemetry (clients, RSSI, interference, Tx power)
Load current AP config (channels, bandwidth, power, roles)
Is a quiet-hours profile (exam/meeting) active now?
YES
NO
Read schedule: room APs, neighbor APs, boost/reduction options
Shape neighbor APs; reduce Tx power/aggressiveness, protect room AP coverage
Boost room APs; increase Tx power, force 20 MHz, stabilize channel
No special profile: run default fast-loop RPM (interference- and load-aware actions)
Inject RF events and operational overrides (maintenance, isolation, freeze)
Apply guardrails and safety checks: KPI bounds, power limits, change budgets
Commit final channel, bandwidth, power settings to APs and log actions
Wait for next control step

Figure 6: Slow loop Architecture

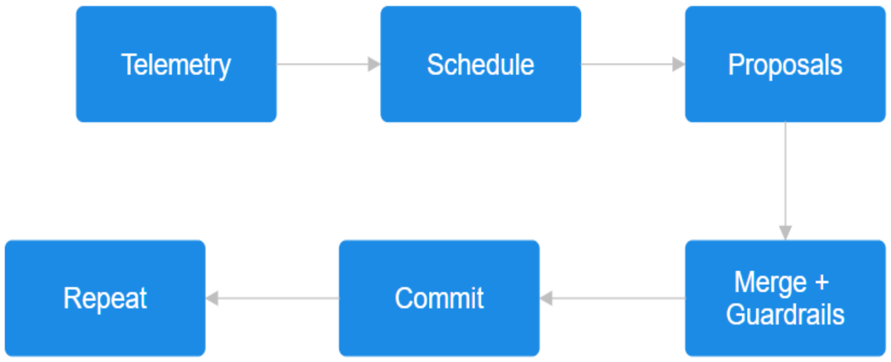


Figure 7: pipeline

7 RRM Controller Design

7.1 High-Level Overview

`rrm_controller_deploy` is a multi-timescale RRM "brain" that sits behind an MQTT broker, continuously ingesting AP telemetry and SDR channel summaries, building an RF-aware interference graph, and then coordinating two control loops: a fast loop that quickly tweaks channel/width on interference spikes, and a slow loop that runs a GNN-based RL controller plus a DSATUR channel planner with strict guardrails and rollback. Telemetry is merged into a shared `TelemetryBuffer`, enriched with RF distances and AP–AP neighbours, converted into a PyTorch Geometric graph, and fed to a `GNNQNetwork` that outputs per-AP actions on power, bandwidth and OBSS-PD. These actions are filtered by a legal-action mask and KPI-based guardrails before being pushed back to APs via MQTT.

7.2 System-Level Architecture

From the code, the overall system can be decomposed into the following components:

- **Wi-Fi Clients**: End hosts generating traffic and QoE metrics (RTT, retries, RSSI). They are not directly instrumented by this script but drive all underlying performance.
- **APs (OpenWrt)**: Each AP periodically publishes telemetry on topics such as `rrm/telemetry/#` and consumes control actions from topics of the form `rrm/actions/{ap_id}`. These actions configure channel, bandwidth, Tx power and OBSS-PD.
- **SDR / Interference Pipeline**: A separate SDR-based scanner publishes interference summaries (e.g., non-WiFi energy, duty cycle, DFS state) on `rrm/summary/logs`. These summaries are mapped to APs and merged into the same `TelemetryBuffer`.
- **MQTT Broker**: Acts as the hub between APs, SDR pipeline and the RRM controller, carrying telemetry, summaries and control actions.
- **RRM Controller (this script)**:
  - *MQTT ingestion* (`on_telemetry_message`, `on_summary_message`, `router_callback`): parses incoming JSON, normalises timestamps and AP IDs, updates the `TelemetryBuffer`, and writes pretty logs with simulated time.
  - *TelemetryBuffer*: Holds a per-AP merged view of recent telemetry and interference, with ageing and snapshot support.
  - *Fast loop worker* (`fast_loop_worker`): runs every few tens of seconds, calls `generate_fastloop_proposals_from_` publishes quick per-AP actions via MQTT and logs all fast-loop decisions.
  - *Slow loop controller* (`slow_loop_controller`): runs every few minutes, executes DSATUR-based channel planning on a conflict graph, enriches the snapshot with client distances and AP neighbours, builds an AP-only interference graph and PyG graph, runs `GNNQNetwork` with epsilon-greedy over a legal action mask, computes reward and guardrail KPIs, optionally triggers rollback, and finally applies actions via MQTT while logging experience tuples.

– *Logging & visualisation*: helpers such as `_log`, `log_experience` and `visualize_rrm_graph_full` generate controller logs, an RL experience dataset and AP+client network visualisations for analysis.

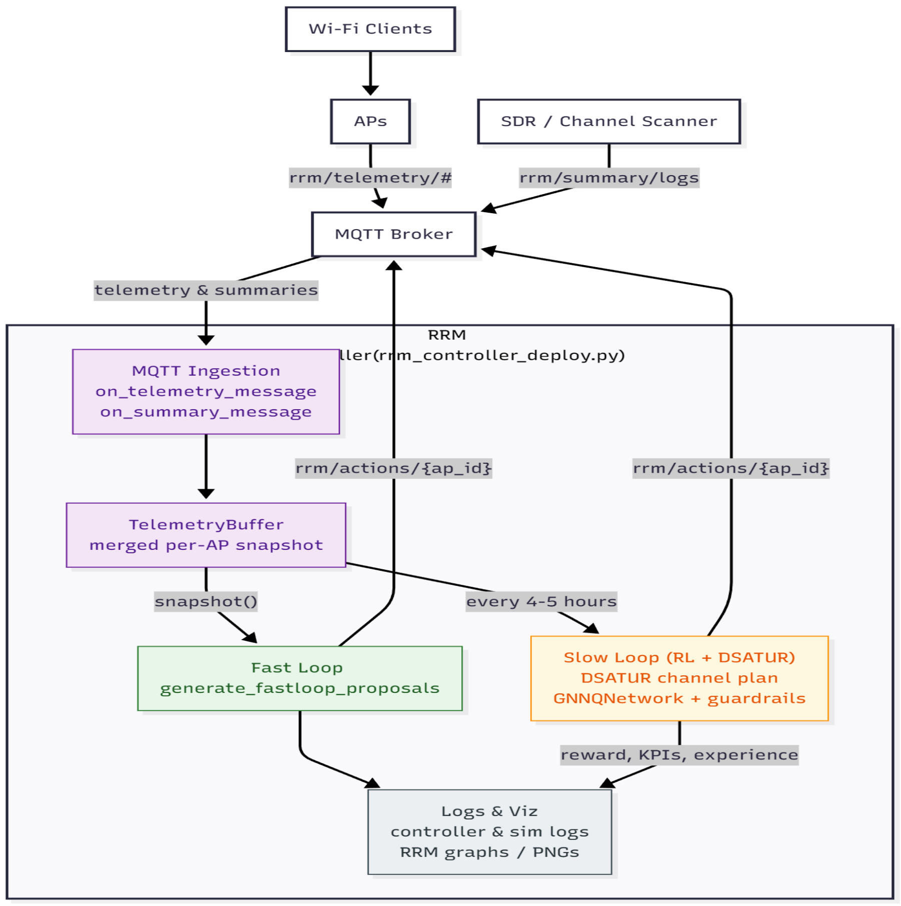


Figure 8: RRM Controller pipeline

7.3 Slow-Loop Control Flow

The `slow_loop_controller` implements a periodic event loop over the global RRM state:

1. **Wait for period & snapshot**: The loop sleeps until the next slow-loop tick based on `slow_period` (e.g., 120 s in pre-train, 5 min in deployment). It then takes a fresh snapshot via `telemetry_buffer.snapshot()`. If the snapshot is empty, it logs "no telemetry", sleeps briefly (5 s), increments `step_idx` and retries.

2. **Channel planning (DSATUR)**: On a non-empty snapshot, the controller builds a dense conflict graph over APs and runs a DSATUR-based planner to propose a channel plan, reusing the previous plan where possible to preserve stickiness.

3. **Visualisation (optional)**: The controller can render a "full RRM graph" with AP+client nodes and channel colouring for debugging and report-quality PNGs.

4. **Graph & feature enrichment**: The snapshot is enriched with per-client RF distances and per-AP aggregates, and then with AP neighbours and co-channel RSSI estimates, giving a rich RF-grounded feature set.

5. **Build RL graph**: An interference graph G_{rl} is constructed (edge weights reflect co-channel overlap, distance and RSSI). This graph is converted into a PyTorch Geometric `Data` object plus an `ap_index_map` for feeding the GNN.

6. **Legal action mask & gating**: A `legal_mask` is built to enforce per-AP change intervals, bandwidth and OBSS-PD bounds, blast-radius limits, peak-hour gating and locality filters. During warm-up or cooldown, all APs can be forced to `NO_OP`.

7. **Action selection (GNNQNetwork)**: The `GNNQNetwork` is evaluated on the graph to produce $Q(s, a)$ per AP. An epsilon-greedy policy is applied over the legal mask to select concrete actions for each AP.

8. **Guardrail KPIs, reward, rollback**: Guardrail KPIs are computed (e.g., P95 RTT, loss, coverage), along with an RL reward and guardrail flags. If any KPI violations are detected, the loop may schedule rollback and restore the last good configuration.

9. **Apply actions & log experience**: If rollback is triggered, a rollback configuration is pushed via MQTT and a `rollback_done` event is logged. Otherwise, the chosen actions are applied via MQTT and the transition (s, a, r, s') plus metadata is appended to `rrm_experience_log.jsonl`.

10. **Sleep & loop**: The loop records elapsed time, sleeps for the remaining portion of `slow_period`, logs a `slow_loop_step_done` event, increments `step_idx`, and then repeats.

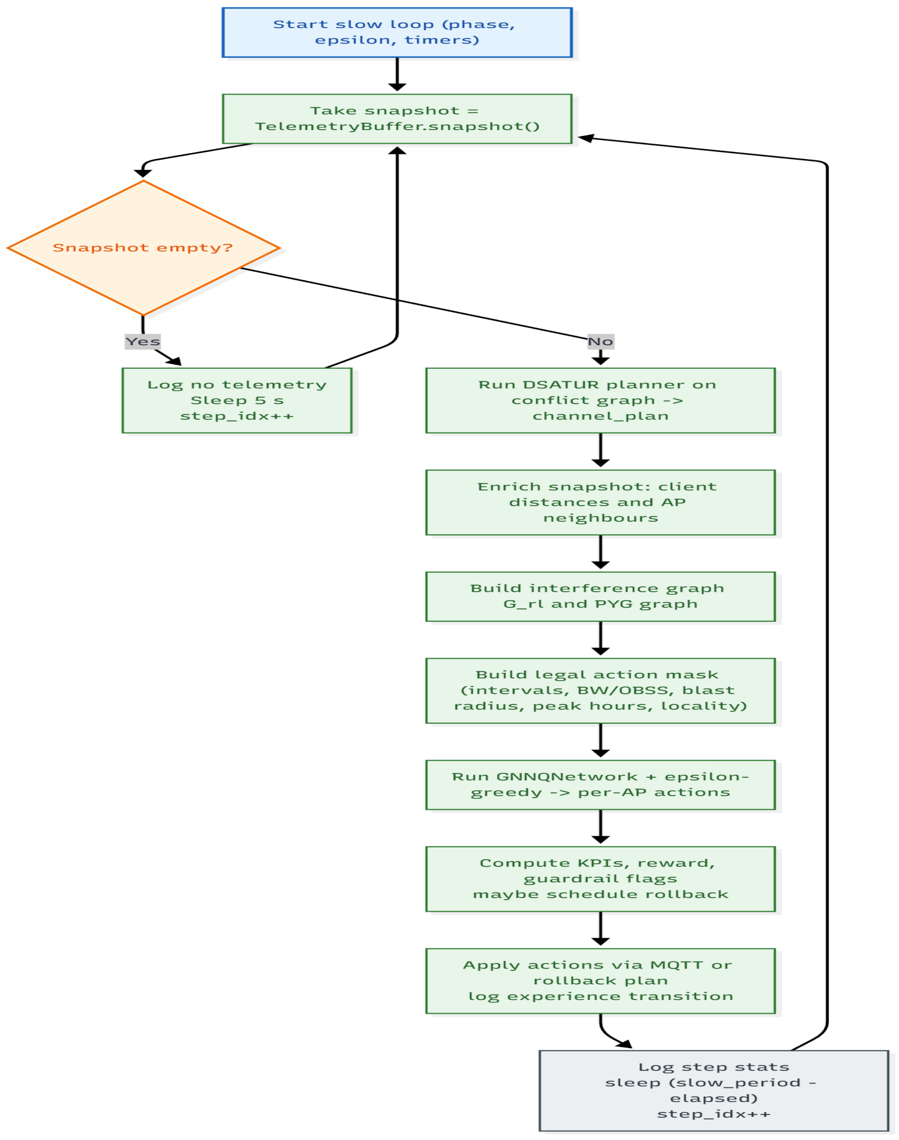

Start slow loop (phase, epsilon, timers)
Take snapshot = TelemetryBuffer.snapshot()
Snapshot empty?
Yes
No
Log no telemetry Sleep 5 s step_idx++
Run DSATUR planner on conflict graph -> channel_plan
Enrich snapshot: client distances and AP neighbours
Build interference graph G_rl and PYG graph
Build legal action mask (intervals, BW/OBSS, blast radius, peak hours, locality)
Run GNNQNetwork + epsilon-greedy -> per-AP actions
Compute KPIs, reward, guardrail flags maybe schedule rollback
Apply actions via MQTT or rollback plan log experience transition
Log step stats sleep (slow_period - elapsed) step_idx++

Figure 9: Slow loop Architecture

7.4 RSSI Calculation and Interference Graph Construction

7.4.1 Friis Transmission Equation for RSSI

The received signal strength (RSSI) between two antennas (AP and client) is given by the Friis transmission equation:

$$\text{RSSI} = \frac{P_t \cdot G_t \cdot G_r \cdot \lambda^2}{(4\pi)^2 \cdot d^2 \cdot L} + P_n + P_i$$

Where:

- P_t = Transmit power (in watts)
- G_t, G_r = Antenna gains (linear)

- λ = Wavelength (in meters)
- d = Distance (in meters)
- L = System loss
- P_n = Noise floor (in watts)
- P_i = Interference power (in watts)

The distance between the AP and client can be derived as:

$$d = \sqrt{\frac{P_t \cdot G_t \cdot G_r \cdot \lambda^2}{(4\pi)^2 \cdot L \cdot (RSSI - P_n - P_i)}}$$

7.4.2 Noise Floor Calculation

The noise floor in watts is calculated from the following formula:

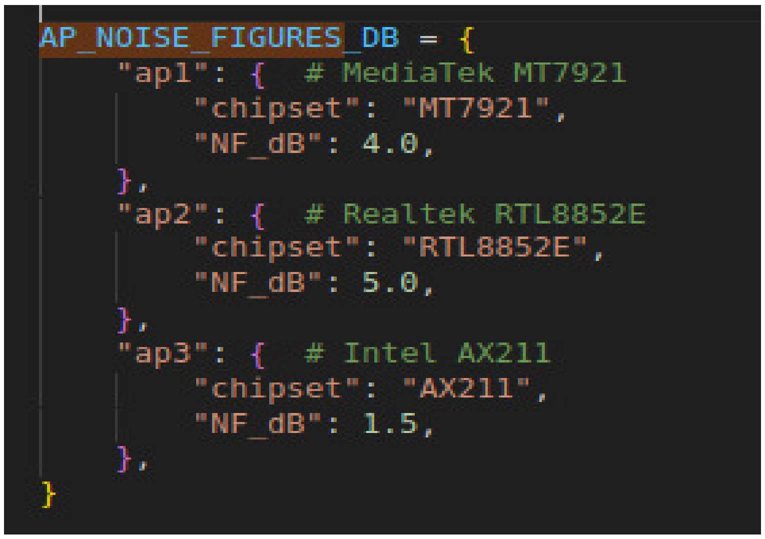
$$N_{\text{floor}}(dBm) = -174 + 10\log_{10}(B) + NF + 10\log_{10}\left(\frac{T}{290}\right) + \Delta NF$$

Where:

- B = Bandwidth (in Hz), derived from wavelength
- NF = Noise figure (in dB)
- T = Temperature (in Kelvin)
- ΔNF = Adjustment factor

The noise floor in watts is then:

$$P_n = 10^{\frac{N_{\text{floor}}}{10}} \times 10^{-3}$$



```
AP_NOISE_FIGURES_DB = {
    "ap1": {  # MediaTek MT7921
        "chipset": "MT7921",
        "NF_dB": 4.0,
    },
    "ap2": {  # Realtek RTL8852E
        "chipset": "RTL8852E",
        "NF_dB": 5.0,
    },
    "ap3": {  # Intel AX211
        "chipset": "AX211",
        "NF_dB": 1.5,
    },
}
```

Figure 10:

7.4.3 Interference Graph Construction

The interference graph is built by calculating the RSSI values for both AP-client and AP-AP relations using the formulas above. The graph edges represent communication links, and the edge weights represent the RSSI, which is affected by both distance and interference.

7.5 Proof of Concept: Real-Time OBSS-PD Emulation

Motivation: While our Slow Loop operates on standard Linux SoftAPs, the Fast Loop OBSS-PD mechanism requires PHY-layer register access that is locked on standard consumer NICs (e.g., Intel AX210). To validate the decision-making logic without custom hardware, we devised a *Digital Twin* emulation strategy.

Logic & "Weakest Client" Constraint: We implemented a controller that constructs a real-time interference graph based on RF coupling rather than static topology. The algorithm calculates the maximum safe OBSS threshold (T_{OBSS}) using the constraint:

$$T_{OBSS} \leq RSSI_{weakest} - \Delta safety\_margin \tag{1}$$

If the detected neighbor RSSI is below T_{OBSS}, the system flags the interaction as **SR ACTIVE** (Green), implying the neighbor can be treated as noise. If above, it is **BLOCKED** (Red), requiring standard backoff.

Emulation Mechanism: We utilized **Linux Traffic Control (tc)** to proxy the physical layer impact. When the algorithm returns *SR ACTIVE*, artificial limits are removed, simulating full airtime utilization. When *BLOCKED*, netem injects latency to mimic the throughput penalty of CSMA/CA contention slots. This confirms that our Fast Loop correctly identifies spatial reuse opportunities in real-time.

8 Advanced Client View

The passive RTT measurement framework is designed to provide a comprehensive view of network performance without relying on active protocols like 802.11mc (FTM). The system consists of multiple interconnected layers that perform packet capture, metric computation, anomaly detection, and data storage, while also incorporating RSSI-based client categorization for spatial analysis.

8.1 Packet Capture Layer

Description: The system continuously captures network packets from a specified wireless interface (e.g., `wlan0`). Using the `Scapy` library, it focuses on TCP/IP packets, specifically those carrying TCP timestamp options (`TSval` and `TSecr`).

Purpose: This layer serves as the data collection point, obtaining raw packet information required to compute RTT values and other relevant metrics.

8.2 RSSI-Based Client Binning Layer

Description: To overcome the lack of 802.11mc support, clients are classified into three distinct RSSI (Received Signal Strength Indicator) bins: Near, Mid, and Edge. This spatial categorization approximates client location in relation to the access point based on signal strength.

- **Near:** Greater than -45 dBm

- **Mid:** Between -45 dBm and -65 dBm
- **Edge:** Between -65 dBm and -75 dBm

Purpose: Provides coarse spatial data (proximity to access points) to complement RTT measurements and help correlate network performance with client location.

8.3 Measurement and Metric Calculation Layer

Description: RTT values are derived by correlating `TSval` in client packets with `TSecr` in corresponding server ACKs. Key performance metrics are calculated per 15-second epoch:

- **Median RTT:** Provides the typical latency.
- **P95 RTT:** Captures the tail latency (95th percentile).
- **RTT Variation (Std Deviation):** Measures the variability in RTT.
- **Packet Loss Rate:** Estimates loss based on retransmissions.
- **Loss Variance:** Quantifies the instability in packet loss over time.

Purpose: Transforms raw network data into actionable insights by quantifying network latency, packet loss, and overall stability for each client.

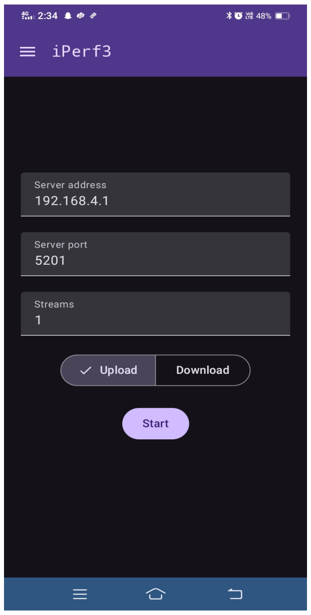
8.4 Anomaly Detection Layer (CUSUM-Based)

Description: The system implements a CUSUM (Cumulative Sum) algorithm to detect sudden increases in RTT, which could indicate congestion, interference, or network disruptions. It applies a small drift to avoid flagging minor fluctuations as anomalies.

Purpose: Monitors for abrupt latency spikes and provides real-time alerts on performance degradation, helping to identify network issues like congestion or coverage problems.

8.5 System Benefits

No Need for 802.11mc Support: The system operates without the need for advanced location-based protocols like 802.11mc, making it suitable for environments where such protocols are unavailable.
Comprehensive Client QoE Profiling: By combining RTT measurements with RSSI-based spatial binning and anomaly detection, the system provides a holistic view of network performance and client experience.
Real-Time Monitoring: The CUSUM-based spike detection ensures that sudden latency increases are identified in real-time, providing immediate insights into potential network issues.



Server address
192.168.4.1
Server port
5201
Streams
1

Some RTT Results:

Samples : 50
RTT Median : 0.26 ms
RTT P95 : 1.62 ms
Loss rate : 3.85 %
Loss variance : 0.036982

Samples : 50
RTT Median : 0.33 ms
RTT P95 : 4.20 ms
Loss rate : 1.96 %
Loss variance : 0.019223

Figure 11: Event loop

9 Conclusion

In conclusion, the H4 “RRM-Plus” problem has pushed us to move from traditional, heuristic AP-centric tuning towards a principled, client-centric, sensing-aware RRM stack that can operate safely in dense, interference-rich enterprise deployments. Through our mid-term work, we have shown that (i) additional-radio sensing combined with robust non-Wi-Fi classification and change detection can provide reliable context for decision-making, (ii) a unified client-view telemetry schema and BSS-transition logic can translate this context into concrete, per-client actions, and (iii) a guarded SafeChangePlanner with BO-based startup, churn budgets and rollback can enforce SLO-driven changes without destabilising the network. The remaining end-term work will focus on tightening the loop between telemetry and control via RL/BO extensions, scaling evaluation across richer topologies and interference mixes, and hardening the USRP/Mininet-WiFi testbed and Linux AP testbed so that RRM-Plus is not just algorithmically sound, but also deployable, interpretable and robust enough to serve as a practical blueprint for Arista’s next-generation RRM.